Available online at www.sciencedirect.com

ScienceDirect

Procedia CIRP 41 (2016) 247 – 251

www.elsevier.com/locate/procedia

48th CIRP Conference on MANUFACTURING SYSTEMS - CIRP CMS 2015

Methodology towards computer-aided testing of complex mechatronic systems: a case study about assembling a train system

David Meinel*, Paryanto, Jörg Franke

Institute for Factory Automation and Production Systems, Friedrich-Alexander-University of Erlangen-Nuremberg (FAU), Egerlandstr. 7-9, Erlangen D-91058, Germany

* Corresponding author. Tel.: +49-9131-8520195; fax: +49-9131-302528. *E-mail address:* david.meinel@faps.fau.de

Abstract

Domain-specific computer-aided analysis of technical systems has become well-established. Nominal and worst case scenario behavior of systems can be modelled sufficiently, however, still lacking the ability of simulating the behavior of technical systems as a whole, combining single-domain subsystems. Therefore, unpredictable failure can occur, which displays the demand for computer-aided system analysis. To counteract the foresaid, a concept has been proposed, which enables sufficient designing and implementing a highly flexible model landscape. It serves the process from comprehending a technical system's specifications and functions towards an executable model structure, using the Modelica-based modelling software SimulationX. Since its future application remains to be tested in the aftermath, the concept will exemplarily be applied on a flexible train simulator.

Peer-review under responsibility of the scientific committee of 48th CIRP Conference on MANUFACTURING SYSTEMS - CIRP CMS 2015

Keywords: Computer-aided testing; Mehatronic system modelling; SimulationX; Train systems; Flexible simulator; Train simulator; Train modelica.

1. Introduction

With regard to sufficient virtual testing of complex mechatronic systems, such as platform-based trains, it is of great importance to build up and maintain a flexible, easily extendable model landscape. Its application has not to rely on advanced developer's knowledge and might run on test engineers' PCs. Therefore, and in order to being accepted by test engineers as well, as becoming well-integrated into a company's tool environment, the required tool knowledge and setting-up time has to be minimized. Moreover, the user interfaces of model landscapes need to be mostly self-explanatory in order to create new subsystems, e.g., entire wagons or even lower-level subsystems, such as coupler or controller models. They are to be designed by following a proposed, modular concept, in which modules can be simply replaced with other modules, just like in Lego brick toys, but virtually and using technical component models as modules.

A major accomplishment is to concept a model landscape, which enables instant setting-up of a ready-to-run simulation scenario, even though it can contain not only variable quantities, but also variable models, such as a variable number of compartments in a train, for instance [1].

1.1. State of the art

Development of mechatronic products always involves engineers from several areas of expertise, including mechanics, electrics, control engineering and software engineering (embedded) [2]. Therefore, a tool as a bridge to communicate product system requirements is needed, which is capable of analyzing product behavior and optimizing product parameters. This is not easy, since engineering experts are differently experienced and have a different background. Simulation and modeling software tools are common tools, being used to solve that issue, since it generates data that every engineer can understand. However, most of these software tools are using their own modeling platform and algorithms. Thus, they cannot be used to support the whole development process. For example, the development of mechatronic control systems is using MATLAB Simulink, while for mechanical and dynamic analysis ANSYS is being used. As authors know, at this

Peer-review under responsibility of the scientific committee of 48th CIRP Conference on MANUFACTURING SYSTEMS - CIRP CMS 2015
doi:10.1016/j.procir.2015.12.121

moment there are not any simulation tools that are capable of being used in the development of a mechatronic system as a whole. Several methods have been proposed by researchers to integrate all or some of these platforms. However, they are still leaking integration methodology.

One of the common solutions for this issue is the conduction of a multi-domain simulation approach, which is capable of simulating all mechatronic system domains, e.g., control engineering, mechanical engineering, pneumatics, hydraulics etc. in one simulation tool environment [3-8].

2. Object-oriented approaches and their benefits

Different enablers towards the aforesaid high flexibility and user-friendliness have been identified, which are the involvement of an object-oriented tool structure, the application of interface classes instead of finalized classes and a flexible implementing concept, implementing the specifications and functions of a technical system. Only after the library class structure is being completed, instancing about the classes inside a top-level model landscape takes place.

In this chapter, the benefits through object-oriented modelling tools will be explained, such as SimulationX, which is advantageous regarding establishing a flexibly configurable model landscape for modern civil railways. On one hand, the methodic implementation of several class generations can be helpful, whereas on the other, the use of interface classes can enable a modular structure.

2.1. Properties of object-oriented inheritance structures in SimulationX

Enabling future extensions and modifications on the physical modelling level inside the compartment class is as important as changing and extending the management and interaction of the compartment interfaces on the train modelling level. Thereby, object-oriented designs and inheritances have turned out useful. Proceeding that way, subsequent changes of essential behaviors, parameters or variables to be done will also cause a change in the inheriting classes and their successors. Since this is taken into account, the distribution effort of parameter sets and model set-ups inside a complex model landscape can be significantly reduced. The mentioned benefits will be explained by an example.

Proposing a top-level class for a train compartment, that is able to simulate kinetic behavior and features interfaces to the surrounding compartments, provides the basis for any sort of compartment, be it about train types or different compartment types in a particular train. Any parameter, variable or behavioral equation being changed in this class will also affect any successor. Assuming two different trains with compartments of different model structures, it is advantageous to create two different successors, containing the different physical model structures. If any essential changes have to be made, concerning parameters, variables and fundamental equations, it is to be done in the top-level class. Both inheriting compartment classes about the 2 different trains, even bound instances about the classes, will obtain these changes. However, if the physical structure of the different train compartments is to be changed, it has to be done in the inheriting classes, as it is meant to affect the train-specific class, only. The example is being illustrated in Fig. 1and Fig. 2.

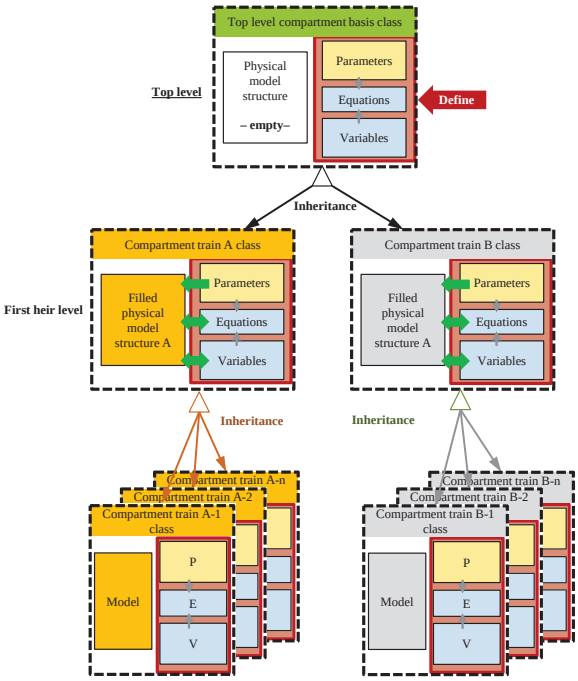

Fig. 1. Inheritance due to top-level class changes.

Fig. 1 shows the inheritance of top-level class modifications about parameters, equations and variables. In Fig. 2, it is being shown, how physical model structure changes in the first heirs level affect all their successors' properties, as well.

The implementation of interface classes allows for variable models inside a static model landscape, in which no structural changes are to be made. Once having set-up an interface class inside the model landscape, which is identical with an ancestor class, during parameterization before a simulation run, any successor can be selected instead. This way, e.g., different compartment types can be quickly changed without interfering into the model structure. Therefore, very little developer knowledge is required, if at all, as changing a component not often demands more than going through a simple drop-down menu. The aforesaid will be exemplarily shown.

As in Fig. 3 illustrated, the declaration of top-level compartment base classes as interface classes enables the user for making the compartments' "placeholders" any of the various heirs available. In this particular example, the compartments 1-4 in the train base class are being implemented, whereas the rest of the compartments remains empty by implementing so called compartment placeholders in order to both featuring a variable number of compartments and keep the model executable.

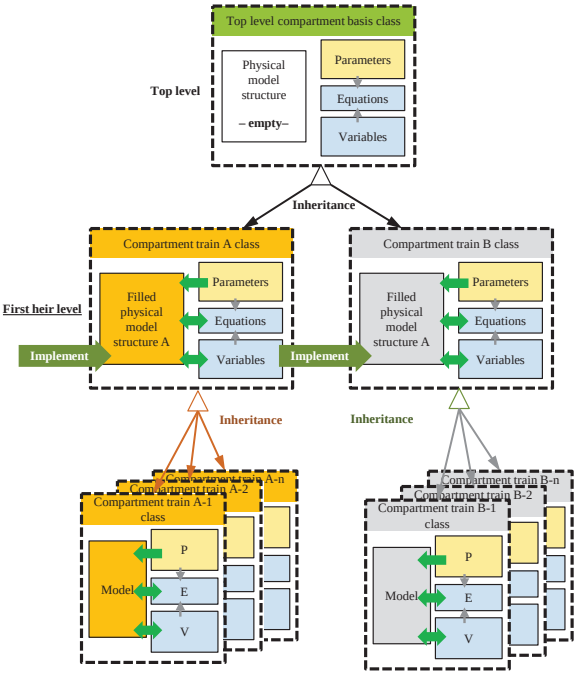

Fig. 2. Inheritance due to 1st heir level class changes.

2.2. Interface classes enabling modular structure

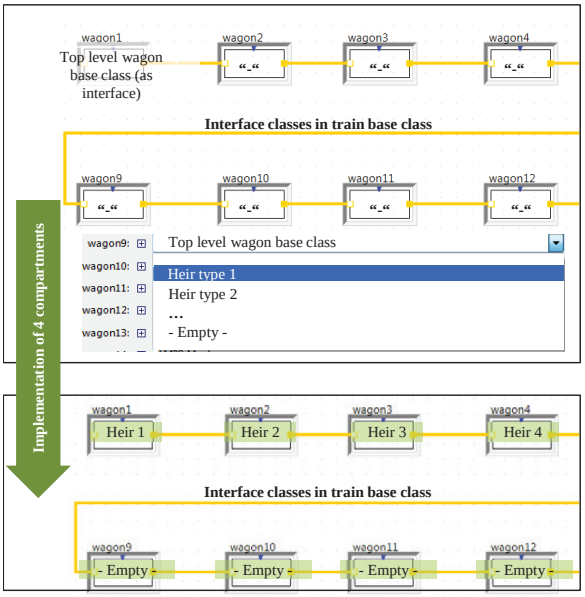

Fig. 3. Implementing interface classes.

3. Flexibility- and object-oriented modelling concept

Since in Chapter 2 the characteristics of object oriented model landscape designs have been introduced, using interface classes, in Chapter 3 a flexibility-oriented designing concept regarding mechatronic, centrally controlled multi-vehicle model landscapes will be proposed.

The concept contains 4 steps from the preliminary identification of the system specification and analysis specification, as illustrated in Fig. 4. The first step consists of all subsystem and overall system functions to be implemented with regard to a running train model, such as relevant physical behavior, relevant environmental effects, relevant controlling behavior, as well using Modelica behavioral equations, application of existing physical models from the library, both physical and logical parameters and variables as well, as analysis-related quantities.

To mention an example for each one of the aforementioned points, one exemplary system function is the capability to move along an axis. A sample physical behavior function would be a braking characteristic, such as the braking force over the braking time. An important environmental effect is the contact physics between the wheels and the rails. As relevant controlling behavior a central controller can be mentioned, being able to represent a central command device, which has the power over all of the compartments.

Applicable developer libraries might be torques, masses or signal processing elements, such as, an integrator. As soon as the parameters, variables and results respectively analysis-related quantities are being identified, designing the top-level class, containing parameters, equations and variables, can take place. Exemplary parameters, variables and result quantities are compartment body mass, axle rotational speed, and compartment speed, whereas result quantities and variables are of the same kind (results), but the identified system-relevant quantities are in this context being distinguished from irrelevant variables and therefore, called analysis-related quantities. A logic parameter is, e.g., position of the current compartment in the train.

As a result, the top-level compartment base class, e.g., can be modelled. What is being defined in this class will be applied in any compartment type, be it, e.g., a tram or a commuter train. Its 1st heir, still lacking any physical model structure, using library models and Modelica-written behavior, inherits all properties – parameters, equations and variables – from its ancestor, which are in this step being extended by a model structure, using the quantities and behavioral equations. In this step, fundamentally different physical structures are to be designed in different sibling classes, thus, the essentially different braking system of, e.g., a tram might not be implementable by using the same model structure as for a commuter train, that usually blends several sorts of physical brakes, such as eddy current brakes, pneumatic brakes, electrodynamic brakes or electromagnetic rail brakes.

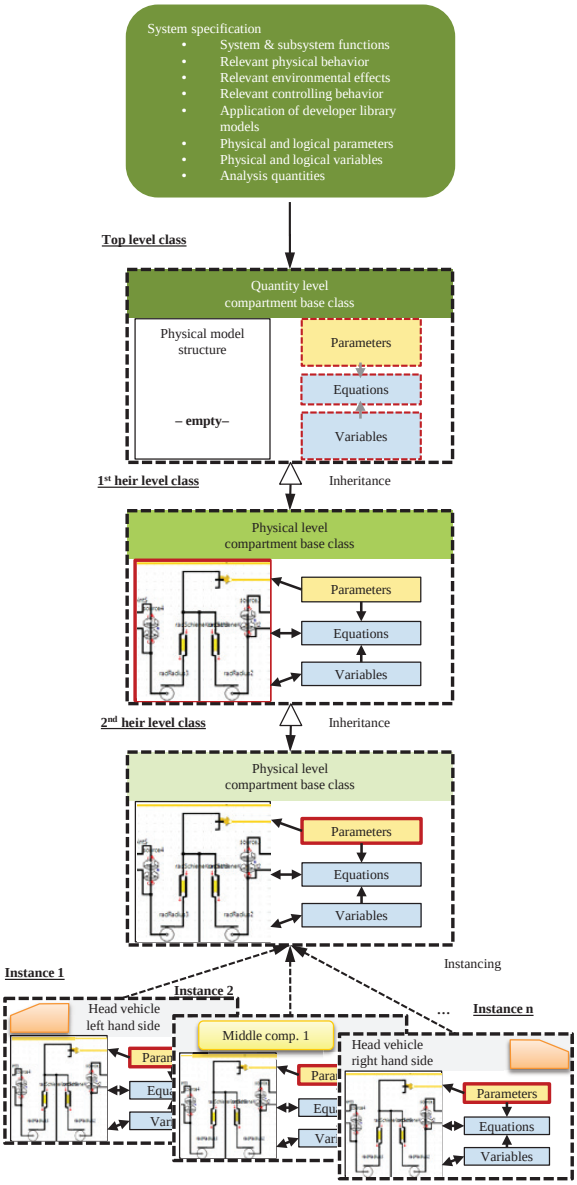

Fig. 4. Illustration of a flexible, object-oriented modelling concept.

The 2nd heir level class embodies a ready-to-use component, such as the model of a particular compartment about a particular train. Inside this class, applied model interface classes are to be implemented. For example, the coupler, having been an interface class so far, is to be implemented towards a ready-to-use heir of the coupler base class, being its ancestor. The further the generation proceeds, the more sibling classes will be created, which can be understood by having a look at the following scenario.

Whereas there is only one top-level class, representing any sort of compartment, a couple of 1st heir level classes can be considered, such as, e.g., trams and commuter rails. Any of them again can be inherited by several 2nd heir level classes, embodying a head or a middle compartment about a particular train. As soon as the 2nd heir level class has been completed, instancing inside the model landscape can begin. Therefore, as many instances from the 2nd heir level classes will be created and bound into the train system.

4. Case study: Modularly assembling a train

Beginning with a plain top-level train base class, all of the compartments' "placeholders" in the model are represented by interface classes, in which the top-level compartment base class is inserted. The next step towards a particular train configuration is the creation of a new class, inheriting all properties from this class. This 1st heir level class will be the instantiable, ready-to-use class. Any number of trains of this type can be instanced from this class. As having the 1st heir created, in its properties the train configuration is to be conducted, in that the implementing, inheriting compartment types will be selected. In here, the 2nd heir level classes about the compartments will be implemented, taking the concept from Chapter 3 into account. One compartment by another can be queued this way, whereas empty interface classes to be are to fill with mock-ups, which represent an object of zero length, breadth, depth and mass. Thus, a flexible number of compartments can be achieved.

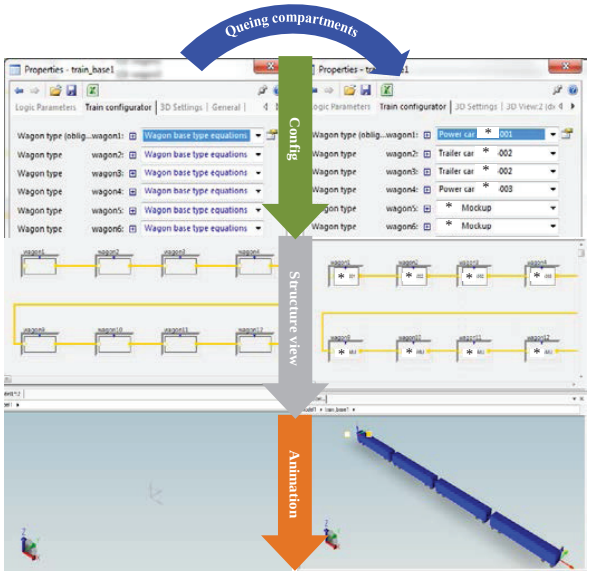

Fig. 5. Queuing wagons in a train class

In Fig. 5 the implementation of a ready-to-use train class is being documented. Beginning with the plain class, step by step compartments are being filled. Whilst on the left side the train does only contain not yet implemented interfaces, step by step compartments are being queued by the user, until the final number and composition of compartments has reached. The rest of the compartments will be completed by using model mock-ups. As soon as the procedure is finished, the fine adjustment of the compartments can be conducted, where after the drive-related parameters can be defined and a simulation be run.

Using this principle, new sorts of trains and compartments can be comparatively quickly created, parameterized and build up without having profound developer knowledge. Moreover, the train classed allows for modifications at any time. Through the 3D representation of the compartments and, thus, the train, users will be able to comprehend the functional principle of the simulator rather fast.

5. Conclusion and Outlook

The object-oriented implementation concept from Chapter 3 has been shown regarding a case study in Chapter 4. Through a 4 steps concept from the specifications towards the applied instances of ready-to-use classes, it is possible to design a model landscape about a complex mechatronic system, such as a train system, whose simulation application requires little developer knowledge. Moreover, by embedding interface classes, subsequent extensions and modifications can also follow, whilst keeping a deeper comprehension about SimulationX away from the user. The presented methodology can also be applied in order to develop a model landscape about manufacturing systems, as the concept is not limited to trains but universally applicable.

In the near future, the virtual train models will be comprehensively tested and applied, regarding the kinetic behavior, especially longitudinal vibrations, due to stimulations by guidance system issues. The investigation efforts and simulation results remain to be published in the afterward.

References

[1] Meinel D, Paryanto, Franke J. Chances of the Application of Multi-Domain Simulation Tools in the Field of Train System Engineering. International FAIM Conference 24th : 2014. Flexible Automation & Intelligent Manufacturing.

[2] Paryanto, Merhof J, Brossog M, Fischer C. An integrated simulation approach to the development of assembly system components. Advanced Materials Research 2013; 769:19-26.

[3] Isermann R. Modeling and design methodology for mechatronic systems. IEEE/ASME Transactions on Mechatronics 1996; 1(1):16- 28. Link: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=491406

[4] Sinha R, Liang V, Paredis C, Khosla P. Modeling and simulation methods for design of engineering systems. Journal of Computing and Information Science in Engineering 2001; 1:84-91.

[5] Barbieri G, Fantuzzi C, Borsari R. A model-based design methodology for the development of mechatronic systems. Mechatronics 2014; 24(7): 833-843.

[6] Komoto H, Tomiyama T. A framework for computer-aided conceptual design and its application to system architecting of mechatronics products. Computer-Aided Design 2012; 44(10): 931-946.

[7] Dieterle W. Mechatronic systems: Automotive applications and modern design methodologies. Annual Reviews in Control 2005; 29(2): 273-277.

[8] Gausemeier J, Dorociak R, Pook S, Nyßen A, Terfloth A. Computer-aided cross-domain modeling of mechatronic systems. Proceedings of International Design Conference 2010; 723-732.